Automated Rubber Sheet Grading: Backend Journey

This guide details the complete setup for the Python Flask backend, designed to serve your pre-trained deep learning model and connect to a MongoDB database.

1. Project Structure

Here is the clean, well-structured directory you should create for your backend:

```
/rubber-grading-backend/
|
|-- /models/
|   |-- .gitkeep  (You will place your rubber_model.h5 file here)
|
|-- /venv/
|   |-- (Python virtual environment files)
|
|-- .env
|-- .gitignore
|-- image_processor.py
|-- model_loader.py
|-- requirements.txt
|-- run.py
```

- `/models/` : This directory is where you **must** place your trained model file (e.g., `rubber_model.h5`).
- `run.py` : This is the main Flask server. It runs the API.
- `model_loader.py` : A helper file to load your model into memory efficiently.
- `image_processor.py` : A helper file to preprocess uploaded images so they match your model's input.
- `requirements.txt` : Lists all the Python libraries we need.
- `.env` : Stores your secret MongoDB connection string.
- `.gitignore` : Ignores files that shouldn't be saved to Git (like `.env` and `venv/`).

2. Step-by-Step Backend Journey

Follow these steps to get your backend running.

Step 1: Set Up MongoDB

1. Go to MongoDB Atlas and create a free account.
2. Create a new **Free Cluster**.
3. Once it's built, click **"Connect"** -> **"Connect your application"**.

4. Select **"Python"** as the driver.
5. Copy the **Connection String** (it looks like `mongodb+srv://...`).
6. You will also need to:
    - Create a **Database User** (e.g., `username: password`).
    - Add your current IP address to the **IP Access List** (MongoDB helps you do this).

Step 2: Set Up Your Project Folder

1. Create the `rubber-grading-backend` directory.
2. Inside it, create your Python virtual environment:

```
python -m venv venv
```

3. Activate the environment:
    - **Windows:** `.\venv\Scripts\activate`
    - **macOS/Linux:** `source venv/bin/activate`
4. Create the `models` directory.
5. Create the `.env` file (see the file I generated for you).
6. Paste your MongoDB Connection String into the `.env` file. **Remember to replace** `<username>`, `<password>`, **and** `<your-cluster-url>` with your actual credentials. I've pre-filled the database name as `rubber_grading_db` for you.

Step 3: Install Dependencies

Install all the required libraries from the `requirements.txt` file I provided:

```
pip install -r requirements.txt
```

Step 4: Add Your Trained Model

This is the part you've already completed!

1. Take your best-performing model file (e.g., `EfficientNetB0.h5`, `ResNet50.h5`).
2. Rename it to `rubber_model.h5`.
3. Place this `rubber_model.h5` file inside the `/models/` directory.

(If you don't want to rename it, just update the `MODEL_PATH` variable in `model_loader.py`).

Step 5: IMPORTANT - Customise the Code

You must check two files to make sure they match your model.

1. `image_processor.py` :
    - I have set the `IMG_SIZE = 224` . This is standard for ResNet, VGG, and EfficientNetB0.
    - **If your model was trained with a different image size (e.g., 299x299), you MUST change the `IMG_SIZE` variable in this file.**
2. `run.py` :
    - Inside the `/api/predict` route, I've written logic to interpret your model's output.
    - I'm assuming your model has **one output neuron** (a `sigmoid` function) where `0` is "Grade B" and `1` is "Grade A".
    - **If your labels are reversed (0=A, 1=B), just swap the names in the `CLASS_NAMES` list at the top of the file.**
    - If your model has **two output neurons** (a `softmax` function), you will need to change `np.round(prediction[0][0])` to `np.argmax(prediction[0])` .

Step 6: Run Your Backend Server!

You are now ready. From your terminal (with the virtual environment still active), simply run:

```
python run.py
```

You should see an output like this, indicating the server is running:

```
* Model loaded successfully.
* Running on [http://127.0.0.1:5000/](http://127.0.0.1:5000/) (Press CTRL+C to quit)
```

Your backend API is now **live** on your computer.

Step 7: How to Test Your API (Optional)

Before connecting the frontend, you can use a tool like **Postman** or **Insomnia** to test your API.

Test 1: Register a User

- **Method:** `POST`
- **URL:** `http://127.0.0.1:5000/api/register_user`
- **Body (JSON):**

```
{
    "name": "Sathwik",
    "location": "Bantwal"
```

```
}
```

- **Result:** You should get a `{"message": "User registered successfully"}`. You can check your MongoDB Atlas dashboard to see the data appear!

Test 2: Get a Prediction

- **Method:** `POST`
- **URL:** `http://1WELCOME.TO.GOTHAM/api/predict`
- **Body (Form-Data):**
  - Set a **key** named `file`.
  - **Change the key type from "Text" to "File".**
  - Upload a sample image of a rubber sheet.
- **Result:** You should get a JSON response almost instantly, as you requested:

```
{
    "grade": "Grade A",
    "confidence": "98.72%"
}
```

Next Steps

Your backend is complete. As you said, the next step is to connect this to Shama's frontend. She will need to make API requests (fetches) from her JavaScript code to the two URLs above (`/api/register_user` and `/api/predict`).

Let me know when you're ready for that step!